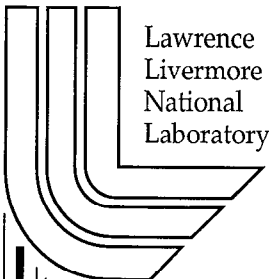


ROSE: Compiler Support for Object-Oriented Frameworks

D. Quinlan

November 17, 1999

U.S. Department of Energy



DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

Work performed under the auspices of the U. S. Department of Energy by the University of California Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

This report has been reproduced
directly from the best available copy.

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information
P.O. Box 62, Oak Ridge, TN 37831
Prices available from (423) 576-8401
<http://apollo.osti.gov/bridge/>

Available to the public from the
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd.,
Springfield, VA 22161
<http://www.ntis.gov/>

OR

Lawrence Livermore National Laboratory
Technical Information Department's Digital Library
<http://www.llnl.gov/tid/Library.html>

ROSE: Compiler Support for Object-Oriented Frameworks

Dan Quinlan¹

Lawrence Livermore National Laboratory, Livermore, CA, USA,
dquinlan@llnl.gov,
WWW home page: <http://www.llnl.gov/CASC/people/quinlan/>

Abstract. ROSE is a preprocessor generation tool for the support of compile time performance optimizations in **Overture**. The **Overture** framework is an object-oriented environment for solving partial differential equations in two and three space dimensions. It is a collection of C++ libraries that enables the use of finite difference and finite volume methods at a level that hides the details of the associated data structures. **Overture** can be used to solve problems in complicated, moving geometries using the method of overlapping grids. It has support for grid generation, difference operators, boundary conditions, database access and graphics. In this paper we briefly present **Overture**, and discuss our approach toward performance within **Overture** and the A++P++ array class abstractions upon which **Overture** depends, this work represents some of the newest work in **Overture**. The results we present show that the abstractions represented within **Overture** and the A++P++ array class library can be used to obtain application codes with performance equivalent to that of optimized C and Fortran 77. ROSE, the preprocessor generation tool, is general in its application to any object-oriented framework or application and is not specific to **Overture**.

1 Introduction

The **Overture** framework is a collection of C++ libraries that provide tools for solving partial differential equations. **Overture** can be used to solve problems in complicated, moving geometries using the method of overlapping grids (also known as overset or Chimera grids). **Overture** includes support for geometry, grid generation, difference operators, boundary conditions, database access and graphics.

An overlapping grid consists of a set of logically rectangular grids that cover a domain and overlap where they meet. This method has been used successfully over the last decade and a half, primarily to solve problems involving fluid flow in complex, often dynamically moving, geometries. Solution values at the overlap are determined by interpolation. The overlapping grid approach is particularly efficient for rapidly generating high-quality grids for moving geometries. As the component grids move only the boundary points to be interpolated change, the grid points do not have to be regenerated. The component grids are structured so

- **Grid functions** [10]: storage of solution values, such as density, velocity, pressure, defined at each point on the grid(s). Grid functions are derived from A++/P++ array objects.
- **Operators** [2,9]: provide discrete representations of differential operators and boundary conditions
- **Grid generation** [12]: the Ogen overlapping grid generator automatically constructs an overlapping grid given the component grids.
- **Plotting** [13]: a high-level interface based on OpenGL allows for plotting **Overture** objects.
- **Adaptive mesh refinement**: The AMR++ library is an object-oriented library providing patch based adaptive mesh refinement capabilities within **Overture**.

Solvers for partial differential equations, such as the **OverBlown** solver are available from the **Overture** Web Site.

2.1 Array Operations

A++ and P++ [17,29] are array class libraries for performing array operations in C++ in serial and parallel environments, respectively.

A++ is a *serial* array class library similar to FORTRAN 90 in syntax, but not requiring any modification to the C++ compiler or language. A++ provides an object-oriented array abstraction specifically well suited to large-scale numerical computation. It provides efficient use of multidimensional array objects which serves to both simplify the development of numerical software and provide a basis for the development of parallel array abstractions. P++ is the *parallel* array class library and shares an identical interface to A++, effectively allowing A++ serial applications to be recompiled using P++ and thus run in parallel. This provides a simple and elegant mechanism that allows serial code to be reused in the parallel environment.

P++ provides a data parallel implementation of the array syntax represented by the A++ array class library. To this extent it shares a lot of commonality with FORTRAN 90 array syntax and the HPF programming model. However, in contrast to HPF, P++ provides a more general mechanism for the distribution of arrays and greater control as required for the multiple grid applications represented by both the overlapping grid model and the adaptive mesh refinement (AMR) model. Additionally, current work is addressing the addition of task parallelism as required for parallel adaptive mesh refinement.

Here is a simple example code segment that solves Poisson's equation in either a serial or parallel environment using the A++/P++ classes. Notice how the Jacobi iteration for the entire array can be written in one statement.

```
// Solve u_xx + u_yy = f by a Jacobi Iteration
Range R(0,n)                // define a range of indices: 0,1,2,...,n
floatArray u(R,R), f(R,R)    // declare two two-dimensional arrays
f = 1.; u = 0.; h = 1./n;    // initialize arrays and parameters
```

```

Range I(1,n-1), J(1,n-1);      // define ranges for the interior

for( int iteration=0; iteration<100; iteration++ )
    u(I,J) = .25*(u(I+1,J)+u(I-1,J)+u(I,J+1)+u(I,J-1)-f(I,J)*(h*h)); // data parallel

```

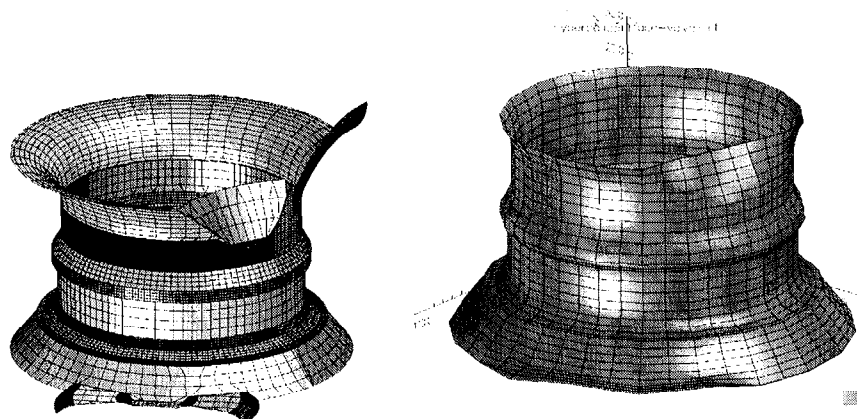


Fig. 1. Hyperbolic surface grid generation is used to generate a smooth surface grid over a surface coming from a CAD package.

2.2 Grid Generation

Overture has support for the creation of overlapping grids for complicated geometries. The process of generating an overlapping grid consists of two basic steps. In the first step a number of component grids are generated. Each component grid represents a portion of the geometry. The component grids must overlap but otherwise can be created locally. **Overture** provides a collection of **Mapping** classes that can be used to generate component grids including splines, NURBS, bodies of revolution, hyperbolic grid generation, elliptic grid generation, trans-finite interpolation and so on. In addition we are working on methods for reading files generated by CAD programs and generating grids. Figure (1) shows how hyperbolic grid surface grid generation can be used to generate a single smooth grid over a CAD surface described by a collection of trimmed NURBS. This is accomplished with the aid of the SURGRD hyperbolic surface grid generator[5].

Given the component grids, the overlapping grid then is constructed using the Ogen grid generator. This latter step consists of determining how the different component grids interpolate from each other, and in removing grid points from holes in the domain, and removing unnecessary grid points in regions of

excess overlap. **Ogen** requires a minimal amount of user input. The grids in figure (2) were all created with **Ogen** and represent some of the new grid generation capabilities within **Overture**.

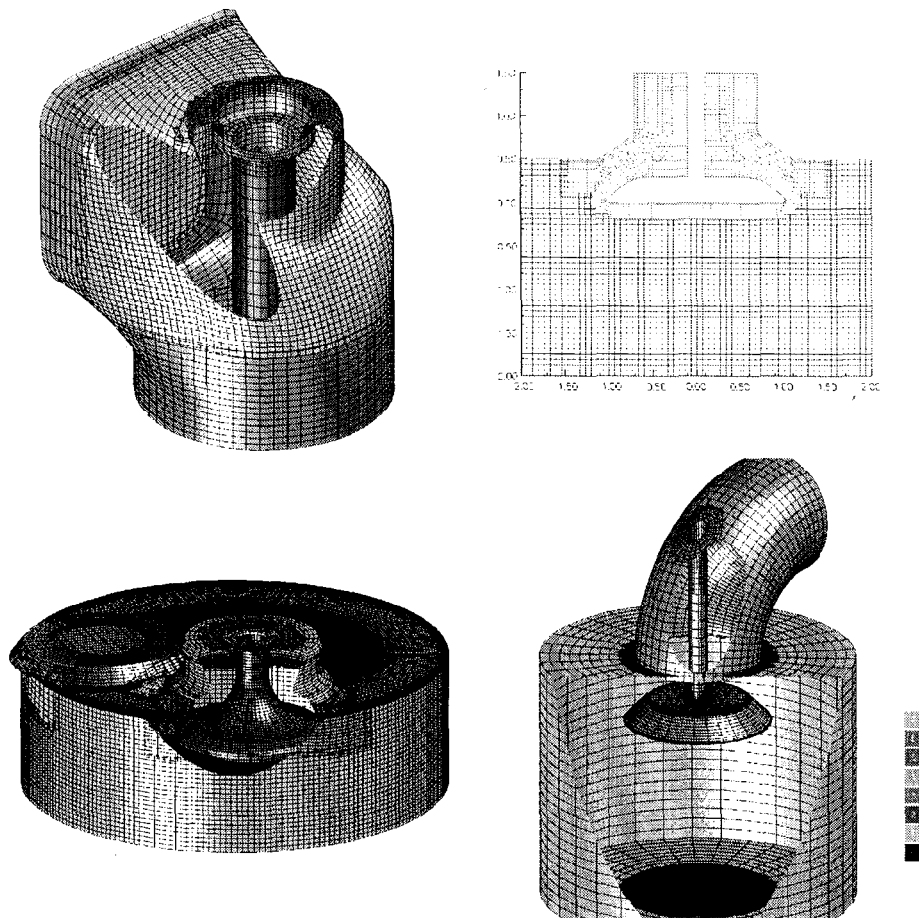


Fig. 2. Sample 2D and 3D overlapping grids generated with the Ogen grid generator.

3 Writing PDE solvers

This example demonstrates the power of the **Overture** framework by showing a basically complete code that solves the partial differential equation (PDE)

$$u_t + au_x + bu_y = \nu(u_{xx} + u_{yy})$$

on an overlapping grid.

The `PlotStuff` object is used to interactively plot contours of the solution at each time step[13].

```
int main()
\{
    CompositeGrid cg;                // create a composite grid
    getFromADataBaseFile(cg,"myGrid.hdf"); // read the grid in
    floatCompositeGridFunction u(cg); // create a grid function
    u=1.;                            // assign initial conditions
    CompositeGridOperators op(cg);    // create operators
    u.setOperators(cg);
    PlotStuff ps;                    // make an object for plotting
    // --- solve a PDE ---
    float t=0, dt=.005, a=1., b=1., nu=.1;
    for( int step=0; step<100; step++ )
    \{
        u+=dt*( -a*u.x()-b*u.y()+nu*(u.xx()+u.yy()) );
        t+=dt;
        u.interpolate();             // interpolate overlapping boundaries
        // apply the BC u=0 on all boundaries
        u.applyBoundaryCondition(0,dirichlet,allBoundaries,0.);
        u.finishBoundaryConditions();
        ps.contour(u);               // plot contours of the solution
    \}
    return 0;
\}
```

The example solves the time-dependent equation explicitly. Other class libraries within the **Overture** framework simplify the solution of elliptic and parabolic equations, the linear systems generated can be solved using any of numerous numerical methods as appropriate including multigrid, and methods made available within a number of external dense and sparse linear algebra packages including PETSc, and others. These are wrapped into the elliptic solver library (Oges) within **Overture**.

4 Approach to Performance

The execution of array statements involves inefficiencies stemming from several sources and the problem has been well documented, by many researchers[31,26,27]. Our approach to performance within **Overture** is to use a preprocessor to introduce optimizing source-to-source transformations. The C++ source-to-source preprocessor is built using ROSE; a tool we have designed and implemented to build application specific preprocessors.

ROSE is a programmable source-to-source transformation tool built on top of SAGE[21] for the optimization of C++ object-oriented frameworks. While we have specific goals for this work within **Overture**, ROSE applies equally well to any other object-oriented framework.

A common problem within object-oriented C++ scientific computing is that the high level semantics of abstractions introduced (e.g. parallel array objects) are ignored by the C++ compiler. Classes and overloaded operators are seen as unoptimizable structures and function calls. Such abstractions can provide for particularly simple development of large scale parallel scientific software, but the lack of optimization greatly affects the performance and utility. Because C++ lacks a mechanism to interact with the compiler, elaborate mechanisms are often implemented within such parallel frameworks to introduce complex template-based and/or runtime optimizations (such as runtime dependence analysis, deferred evaluation, runtime code generation, etc.). These approaches are however not satisfactory since they are often marginally effective, require long compile times, and/or are not sufficiently robust.

Preprocessors built using ROSE have a few features that stand out:

1. A hierarchy of grammars are specified as input to ROSE to build (tailor) the preprocessor specific to a given object-oriented application, library, or framework. ROSETTA, a code generator we have designed and implemented, is used to generate an implementation of the grammars that are used internally. The hierarchy of grammars (and their implementations) are used to construct separate program trees internally, one program tree per grammar, each representing the user's application. The program trees are edited as required to replace selected subtrees with other subtrees representing a specific transformation. Quite complex criteria may be used to identify where transformations may be applied, this mechanism is superior to pattern-recognition of static subtrees within the program tree because it is more general and readily tailored.
2. Transformations are specified which are then built into the user application automatically where appropriate. The mechanism is designed to permit the automated introduction of particularly complex transformations (such as the cache based transformations specified in [28], space does not permit an elaboration of this).
3. To simplify the debugging, preprocessor's output (C++ code) is formatted identical to the input application code (except for transformations that are introduced, which have a default formatting). Numerous options are included to tailor the formatting of the output code and to simplify working with either its view directly within the debugger or its reference to the original application source within the debugger. Comments and all C preprocessor (cpp) control structures are preserved within the output C++ code.
4. The design of ROSE is simplified by leveraging both SAGE 2 and the EDG[22] C++ front-end. EDG supplies numerous vendors with the C++ front-end for their compiler and represents the current best implementation of C++. In principle this permits the preprocessors built by ROSE to address the complete C++ language (as implemented by the best available front-end). Modifications have been made to SAGE 2 to permit portability and allow us to fulfill on a complete representation of the language. By design, we leverage many low-level optimizations provided within modern

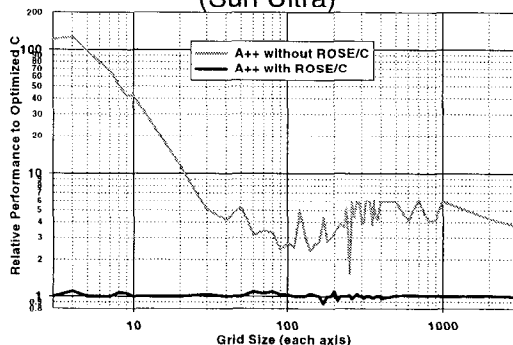
compilers while focusing on higher level optimizations largely out of reach because traditional approaches can not leverage the semantics of high level abstractions. In doing so, we slightly blur the distinction between a library or framework, a language, and a compiler. But, because we leverage several good quality tools the implementation is greatly simplified.

The approach is different from other open C++ compiler approaches because it provides a mechanism for defining high level grammars specific to an object-oriented framework and a relatively simple approach to the specification of large and complex transformations. A requirement for representing the program tree within different user defined grammars is to have access to the full program tree, this is not possible (as we best understand) within the OpenC++[20] research work. By using SAGE 2 and ROSE the entire program tree, represented in each grammar, is made available; this permits more sophisticated program analysis (when combined with the greater semantic knowledge of object-oriented abstractions) and more complex transformations. We believe that the techniques we have developed greatly complement the approaches represented within OpenC++, in particular the Meta object mechanism represented within that work. That SAGE is in many ways similar to the MPC++[19] work, we believe we could have alternatively built off of that tool in place of SAGE (though this is not clear). However, since SAGE 2 uses the EDG front-end we expect this will simplify access to the complete C++ language. MPC++ addresses more of the issues associated with easily introducing some transformations than SAGE, but not of the complexity that we require for cache based transformations[28]. Each represent only a single grammar (the C++ grammar) and this is far too complex (we believe) a starting point for the identification of where sophisticated transformations can be introduced. The overall compile-time optimization goals are related to ideas put forward by Ian Angus[25], but with numerous distinguishing points:

1. We have decoupled the optimization from the back-end compiler to simplify the design.
2. We have developed hierarchies of grammars to permit arbitrarily high level abstractions to be represented with the greatest simplicity with the program tree. The use of multiple program trees (one for each grammar) serves to organize high level transformations.
3. We provide a simple mechanism to implement transformations.
4. We leverage the semantics of the abstractions to drive optimizations.
5. We have implemented and demonstrated the preprocessor approach on several large numerical applications.

Finally, because ROSE is based ultimately (through SAGE) upon the EDG C++ front-end, the full language is made available; consistent with the best of the commercial vendor C++ compilers which most often use the same EDG C++ front-end internally. However, some aspects of the complete support of C++ within SAGE are incomplete.

A++ Performance with and without ROSE
(Sun Ultra)



A++ Performance with and without ROSE
(DEC Alpha)

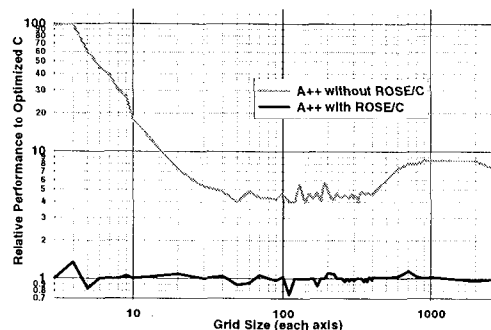


Fig. 3. The use of a preprocessor (built using ROSE) can overcome the performance degradation associated with binary evaluation of array operands. These results show the use of ROSE with A++ and how the performance matches that of optimized C code using the `restrict` keyword (ratio = 1). It has been shown previously that this is equal to Fortran 77 performance. More sophisticated cache-based transformations are also possible.

5 Results

Within our results we consider the following trivial five-point stencil:

$$A(I,J) = c * (B(I-1,J) + B(I+1,J) + B(I,J) + B(I,J+1) + B(I,J-1));$$

In this code fragment, A and B are multidimensional array objects (distributed across multiple processors if P++ is used). In this example, I and J are `Range` objects that together specify an index space of the arrays A and B.

Figure 3 shows the range of performance associated with different size arrays for the simple five point stencil operator on the Sun Ultra and Dec Alpha machines. The Sun Ultra was selected because it is a commonly available computer, the Dec Alpha was selected because its cache design is particularly aggressive and as a result it exemplifies the hardest machine to get good cache performance. The results are in no way specific to this statement, moderate size applications have been processed using preprocessors built with ROSE. The results compare the ratios of A++ performance with and without the use of the ROSE preprocessor to that of optimized C code. The optimized C code takes full advantage of the bases of the arrays being identical and the unit strides, the A++ implementation does not, these very general subscript computations within the array class implementation are compared to very specific and highly optimized subscript computations within the C code. This exaggerates the poorer performance of the A++ statements, we do this to make clear that the performance of the

code output from the ROSE preprocessor is in fact highly optimized and is made specific to the common bases of the operands (determined at compile time) and the unit stride (determined at runtime). Our results show the relative difference that it makes to compare such results. The resulting performance using ROSE is nearly identical to that of the optimized C code (ratio = 1), this is not surprising since the preprocessor transformation replaces the array statement with the equivalent C code (highly optimized, and using restrict pointers where they are supported).

A++ supports expression templates but this data is not presented here, in general the expression template will approach the C performance within 90% for short expressions and sufficiently large arrays. The combination of expression templates with deferred evaluation reduces this to approx. 70% as reported in [27] likely because of the required extra level of indirection to the data required by the deferred evaluation mechanism (it is not clear if this will be fixed)¹.

An important distinguishing point between the two approaches is that within larger applications the compile times are several orders of magnitude less for the preprocessor approach since expression templates are not used[31]. In practice the time taken to pre-process an application is even much less than the compile time where no templates are used (expression templates or otherwise) (a few seconds, and is not noticeable). This is not surprising since the preprocessing consists of only a few of the steps taken internally within a compiler, and excludes the most time consuming back-end optimization (to build the object code).

6 Conclusions

Overture is capable of addressing the complexity of numerous difficult sorts of simulations within scientific computing. We have demonstrated that powerful abstractions can be developed that can greatly simplify the development of previously complex (intractable) simulations. New features within the grid generator in **Overture** have made much more complicated grids possible. While the abstractions presenting within **Overture** are the principle motivation for its use, the performance of **Overture** is critical and is dominated by the performance of the A++P++ array class. Many years of work have gone into the development of optimization techniques for the array class library. The preprocessor approach is by far the most successful so far, however more work remains to make preprocessors easier to build and more robust.

The results we have presented demonstrate the optimization of array class statements. All sizes of arrays benefit, their processing with ROSE makes each equivalent to the performance of optimized C code (using restrict). Previously in [30] we showed that this is equivalent to FORTRAN 77 performance.

Expression Templates is an alternative mechanism that can be used to optimize array statements, but the mechanism is problematic[31]. More research

¹ This was the experience with expression templates when it was combined with the deferred evaluation mechanism in A++P++.

is required (and being done by others) to address problems within the expression template mechanism. More work is similarly required to provide improved compile-time optimization solutions.

7 Software Availability

The **Overture** framework and documentation is available for public distribution from the web site, <http://www.llnl.gov/casc/Overture>. The **OverBlown** flow solver is also available.

References

1. M. J. Berger and P. Colella, *Local adaptive mesh refinement for shock hydrodynamics*, J. Comp. Phys., 82 (1989), pp. 64–84.
2. ———, *Classes for finite volume operators and projection operators*, LANL unclassified report 96-3470, Los Alamos National Laboratory, 1996.
3. D. L. Brown, Geoffrey S. Chesshire, William D. Henshaw and Daniel J. Quinlan, **Overture** : *An Object Oriented Software System for Solving Partial Differential Equations in Serial and Parallel Environments*, Proceedings of the Eight SIAM Conference on Parallel Processing for Scientific Computing, 1997.
4. , D. L. Brown, William D. Henshaw and Daniel J. Quinlan, **Overture** : *An Object Oriented Framework for Solving Partial Differential Equations*, Scientific Computing in Object-Oriented Parallel Environments, Springer Lecture Notes in Computer Science, 1343, 1997.
5. W.M. Chan and P.G. Buning, *A Hyperbolic Surface Grid Generation Scheme and Its Applications*, AIAA paper 94-2208, 1994.
6. G. Chesshire and W. D. Henshaw, *Composite overlapping meshes for the solution of partial differential equations*, J. Comp. Phys., 90 (1990), pp. 1–64.
7. G. S. Chesshire, **Overture** : *the grid classes*, LANL unclassified report 96-3708, Los Alamos National Laboratory, 1996.
8. Diffpack homepage, <http://www.nobjects.com/diffpack>.
9. ———, *Finite difference operators and boundary conditions for Overture, user guide, version 1.00*, LANL unclassified report 96-3467, Los Alamos National Laboratory, 1996.
10. ———, *Grid, GridFunction and Interpolant classes for Overture* , *AMR++ and CMPGRD, user guide, version 1.00*, LANL unclassified report 96-3464, Los Alamos National Laboratory, 1996.
11. ———, *Mappings for Overture* : *A description of the mapping class and documentation for many useful mappings*, LANL unclassified report 96-3469, Los Alamos National Laboratory, 1996.
12. ———, *Ogen: an overlapping grid generator for Overture*, LANL unclassified report 96-3466, Los Alamos National Laboratory, 1996.
13. ———, *PlotStuff: a class for plotting stuff from Overture* , LANL unclassified report 96-3893, Los Alamos National Laboratory, 1996.
14. Satish Balay, William Gropp, Lois Curfman McInnes and Barry Smith, *The Portable Extensible Toolkit for Scientific Computation*, <http://www.mcs.anl.gov/petsc/petsc.html>.

15. Steve Karmesin et.al, *Parallel Object Oriented Methods and Applications*, <http://www.acl.lanl.gov/PoomaFramework>.
16. D. Quinlan, *Adaptive Mesh Refinement for Distributed Parallel Processors*, PhD thesis, University of Colorado, Denver, June 1993.
17. ———, *A++/P++ manual*, LANL Unclassified Report 95-3273, Los Alamos National Laboratory, 1995.
18. Xabier Garaizar, Richard Hornung and Scott Kohn, Structured Adaptive Mesh Refinement Applications Infrastructure, <http://www.llnl.gov/casc/SAMRAI>.
19. Ishkawa et. al. *Design and Implementation of Metalevel Architecture in C++ - MPC++ Approach* -. In *Proceeding of Reflection'96 Conference*, April 1996 more info available at: <http://pdswww.rwcp.or.jp/mpc++/mpc++.html>
20. Shigeru Chiba *Macro Processing in Object-Oriented Languages* In Proc. of Technology of Object-Oriented Languages and Systems (TOOLS Pacific '98), Australia, November, IEEE Press, 1998. more info available at: <http://www.hlla.is.tsukuba.ac.jp/chiba/openc++.html>
21. B. Francois et. al. *Sage++: An object-oriented toolkit and class library for building fortran and c++ restructuring tools*. In *Proceedings of the Second Annual Object-Oriented Numerics Conference*, 1994.
22. Edison Design Group <http://www.edg.com>
23. Rebecca Parsons and Dan Quinlan. *A++/P++ array classes for architecture independent finite difference computations*. In *Proceedings of the Second Annual Object-Oriented Numerics Conference (OONSKI'94)*, April 1994.
24. Dan Quinlan and Rebecca Parsons. *Run-time recognition of task parallelism within the P++ parallel array class library*. In *Proceedings of the Conference on Parallel Scalable Libraries*, 1993.
25. Ian Angus *Applications Demand Class-Specific Optimizations: The C++ Compiler Can Do More*. In *Proceedings of the Object-Oriented Numerics Conference*, (OONSKI) 1993
26. Todd Veldhuizen *Arrays in Blitz++* In *Proceeding of the Second International Symposium, ISCOPE 98*, Santa Fe, NM December 1998
27. Karmesin, et al. *Array Design and Expression Evaluation in POOMA II*. In *Proceeding of the Second International Symposium, ISCOPE 98*, Santa Fe, NM December 1998
28. Bassetti, F., Davis, K., Quinlan, D. *Optimizing Transformations of Stencil Operations for Parallel Object-Oriented Scientific Frameworks on Cache-Based Architectures* In *Proceedings of the ISCOPE'98 Conference*, Santa Fe, New Mexico, Dec 13-16 1998
29. Lemke, M., Quinlan, D., *P++, a C++ Virtual Shared Grids Based Programming Environment for Architecture-Independent Development of Structured Grid Applications* In *Proceedings of the CONPAR/VAPP V*, September 1992, Lyon, France; published in *Lecture Notes in Computer Science*, Springer Verlag, September 1992.
30. Bassetti, F., Davis, K., Quinlan, D. *Toward FORTRAN 77 Performance From Object-Oriented C++ Scientific Frameworks* In *Proceedings of the HPC'98 Conference*, Boston, Mass. April 5-9, 1998
31. Bassetti, F., Davis, K., Quinlan, D. *A Comparison of Performance-enhancing Strategies for Parallel Numerical Object-Oriented Frameworks* In *Proceedings of the first International Scientific Computing in Object-Oriented Parallel Environments (ISCOPE) Conference*, Marina del Rey, California, Dec, 1997